Web Information Systems
Using & developping SOAP services

Lecturer: Stijn Vansummeren

2013–2014

Important comments:

- Skeleton code for all of the exercises below is available on the course website. Download this code before beginning the exercises, extract it, and import it into Eclipse as follows:
  - Choose `File > Import > General > Existing Projects Into Workspace`
  - Select the folder containing all skeletons as the "root folder" - Eclipse will recognize all projects in this folder (be sure to check the "import recursive" option). Click "Finish" to import them.
- This set of exercises is best solved in small groups (2-3 students)

Part I: Consuming SOAP Services

In this first part, we will investigate how to call existing SOAP services from within Java using the Java API for XML Web Services (`jax-ws` for short) API.

In particular, we will use the `wsimport` tool[1] to translate wsdl service definitions into corresponding java code.

Exercise 1.1

In this first exercise, we will use the DailyXmlFact service available at

`http://www.xmlme.com/WSDailyXml.asmx?WSDL`.

1. Open the wsdl file (e.g. by opening the url `http://www.xmlme.com/WSDailyXml.asmx?WSDL` in a web browser). What Service does it define? Which Ports? Which Messages? Which Input/Output Types?
2. Using the `wsimport` tool, generate the caller code for the given service.

   The general syntax to do this is:

   ```
   wsimport -d <class-directory> \
            -s <java-directory> \
            -p <package> \
            http://www.xmlme.com/WSDailyXml.asmx?WSDL
   ```

   Here, `<class-directory>` is the folder where compiled versions of the generated classes will be put; `<java-directory>` is the folder where the generated classes are to be put, and `<package>` is the package to which the generated classes should be put. (It is recommended to take `<package>` = `com.xmlme.dailyxml`.) *Note that the `<class-directory>` and `<java-directory>` folders must exist prior to execution of this command otherwise you will get an error!*

[1] `http://docs.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html`

3. Now compare your responses to question (1) with the generated code. What class is generated to represent the service? What classes to generate the ports? What classes to represent the Input/Ouptut types?

4. Next, we are going to write a program that uses the generated caller code to retrieve the daily fact.

   The general way to do this is as follow. Given a `Service` class that is generated by the `wsimport`.

   ```
   Service myService = new Service();
   System.out.println(
       myService
           .getPort()
           .method("arg1", "arg2"));
   ```

   Here you will need to replace `getPort()` and `method` by the correct port and method generated by `wsimport`.

5. Run your program. What do you get?

6. To get more insight into the messages that are exchanged between your client program and the server, we are going to use the SOAP debugger that is provided in exercise template. To enable this debugger, you need to add the analogue of the following code.

   ```
   Service myService = new Service();
   MyHandlerResolver<SOAPMessageContext> myHandler = new MyHandlerResolver<SOAPMessageContext>();
   myHandler.registerHandler(new SOAPDebugger());
   myService.setHandlerResolver(myHandler);
   ```

   (The `MyHandlerResolver` class will ensure that the SOAP debugger is called.) Like before you should replace `Service` with the corresponding class name generated by the `wsimport` tool.

7. Now run the program again. What are the messages that are exchanged between the client and the service?

Part II: Designing SOAP Services

In the first part, we used existing SOAP services. We now design and implement new SOAP services, again using the Java API for XML Web Services (`jax-ws` for short) API. The framework that we will use to actually implement this API is called Apache CXF. It implements JAX-WS (SOAP) and provides automatic serialization to XML through JAXB. Services can be created Java-first or WSDL-first. A copy of Apache CXF is provided in the lib directory of the corresponding code skeleton (called `RandomExercise`).

Exercise 1.2

We will first implement a service that provides random quotes, and random numbers to its clients.

- The `Random` interface provides a description of the service. JAX-WS uses Java annotations to declare that instances of an interface will be a SOAP-Based web service. Look at the `Random` interface how easily this is done. (Which annotation should you use?)

- The `Random` interface will be used by both the service clients and the service server. The clients simply employ the CXF framework, point it to the HTTP address where the server should run, and ask the framework to return it an implementation of the `Random` interface. All calls to this implementation will be forwarded to the implementation that actually runs on the server. Have a look at the `ServiceClient` class to see how a client is typically constructed.

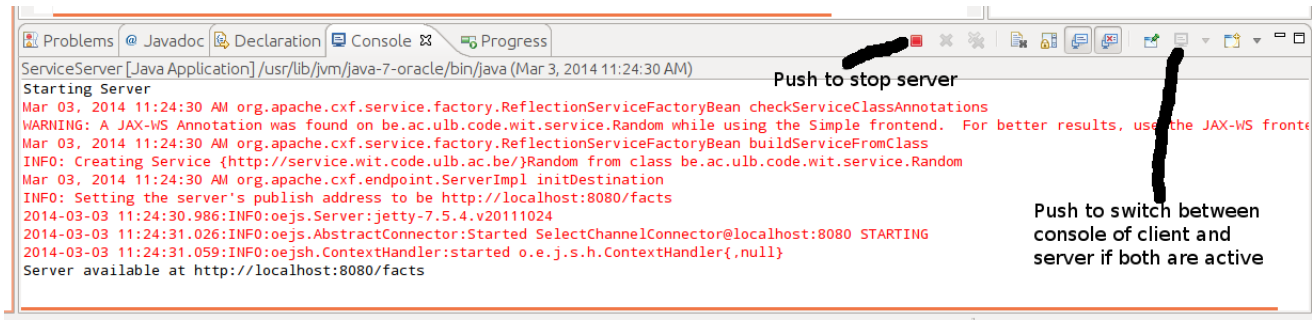

Figure 1: Stopping the server

- In contrast to the clients, which only need the `Random` interface to connect to a server, the actual server itself needs to provide an implementation of the `Random` interface to respond to the web service calls. An empty implementation of the `Random` interface, called `RandomImpl` is provided in the code skeleton. Have a look at the `ServiceServer` class to see how a server is typically set up. This class contains one TODO: you need to register the `RandomImpl` with the server, so that it can use it to respond to web service calls.
- Finally, you need to write the corresponding implementation of the service. The utility class `Quotes` already contain a few quotes. You can use the `ServiceClient` to test your implementation.

  Important:

  - When testing, be sure to first run the ServiceServer by right-clicking on the `ServiceServer.java` file in Eclipse and choosing "Run as">"Java Application". Note that the server keeps running until you terminate it. You can terminate it by clicking on the red button in the Console. (See Figure 1.
  - Next start the client by right-clicking on the `ServiceClient.java` file in Eclipse and choosing "Run as">"Java Application".
  - If both client and server are active, you can switch between the two consoles as indicated in Figure 1.

Exercise 1.3

In the following exercises, we will implement a SOAP-based contact manager service. In this first part, you will define an interface that describes the service. To help you in this task consider the following scenarios and identify the methods that will be called.

- A new contact is created for "Scott Montgomery", with the email address `scott.montgomery@starfleet.gxy`.
- The "USS Enterprise" group is rechristened "USS Enterprise-A".
- "Matt Decker" is removed from the address book.
- The email address "kahless@klingon-empire.gxy" is changed to "kahless-the-unforgettable@klingon-empire.gxy".
- "hiraku.sulu@starfleet.gxy" is removed from the "USS Enterprise" group and added to the "USS Excelsior" group.
- The list of all contacts is retrieved.
- The list of all contacts pertaining to the "Starfleet" group is retrieved.

Exercise 1.4

The `SOAPContactManagerExercise` contains skeleton code that has the necessary support for JAX-WS and, similarly to exercise 1.2, contains code to instantiate the web service server and its clients. (Note that you will need to complete the server and client code). The skeleton also implements a simple model for the address book

and the address book groups. (This model simulates a database, and it specifies how instances of the model can be serialized.)

In this exercise, you are asked to:

- complete the `ContactService` interface (in the `be.ac.ulb.code.wit.service` package) to correspond to the part of your service definition from Exercise 1.3. (Note that this interface only deals with contacts, not with contact groups.).
- Next, you should complete the corresponding implementation `ContactServiceImpl` to provide an actual implementation for your contact manager service without support for contact groups.
- Register the implementation with the server in the `ServiceServer` class.
- Test the implemented methods by completing the client skeleton code in the `ServiceClient` class.

Exercise 1.5

Supplemental: Add support for contact groups management to your service by completing the `GroupService` interface and corresponding `GroupServiceImpl` implementation. (Be sure to also register them with the server and client.)